# Lossy Image Compression by Vector Quantization (VQ)

Presidency University, Kolkata

Ankan Chakraborty

20th November, 2022

## 1 Introduction and methodologies

**K-means** is often called **"Lloyd's algorithm"** in computer science and engineering, and is used in vector quantization for compression. The Kmeans clustering algorithm represents a key tool in the apparently unrelated area of image and signal compression, particularly in vector quantization or VQ. There are three stages to image vector quantization:

### ► Stage - 1: (Prototype Creation)

Generally an image of mn pixels is represented in m rows and n columns. But here first we create arrays by grouping together q local pixels by row (that is q adjacent pixels row-wise). Then we have a collection of vectors $V = \{V_i, i = 1, 2, 3, \ldots, \frac{mn}{q}\}$ where each $V_i$ is a q dimensional vector. This V is called index and the elements $V_i$ are called **index entry**. Some of these vectors, $V_i \in V$, will be similar in some respects. For example, many vectors may be extracted from a uniform area of an image. This leads to the following idea.

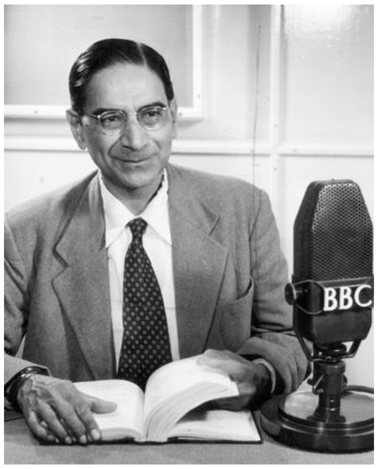
Figure 1: Grayscale image of great statisticisan P.C. Mahalonobis.

- Cluster vectors in $V$ into r groups where $(r \ll \frac{mn}{q})$. The smaller the choice of $r$, the greater is the compression.
- In other words, the set of vectors, $V$, is partitioned into $r$ distinct sets, $S_1, S_2, \ldots, S_r$ corresponding to each cluster.
- Each subset $S_i$ is then been represented by a suitable representative/ prototype vector (may be the cluster mean/mediod). The set of prototype vectors constitutes the **codebook**. Each of the member in the **codebook** is called **codeword** .
- Thus codebook contains $r$ vectors, each of which has an address, $(1, 2, \ldots, r)$.

Because the number of selected elements is much smaller than the number of vectors in the image, the number of bits required to represent the address of a prototype vector is much smaller than the number of bits required to represent an image vector.

In this project we will work with the grayscale image of great statistician P.C. Mahalonobis **(Figure 1)**. Here our image is of $430 \times 346$ pixel resolution i.e. a vector of total $430 \times 346 = 148780$ elements after the image array being flattened into a vector for further operation. And here $m = 430$, $n = 346$.

Lets first read the image in R and a function in R (which takes as input a vector of observations) to plot the original image in R **(Figure 2)**. For this we need ``jpeg'' package to be installed in order to read JPEG images in R.

Now first implement a version of **vector quantization (VQ)** with the choice of $q = 4$ and $r = 15$.

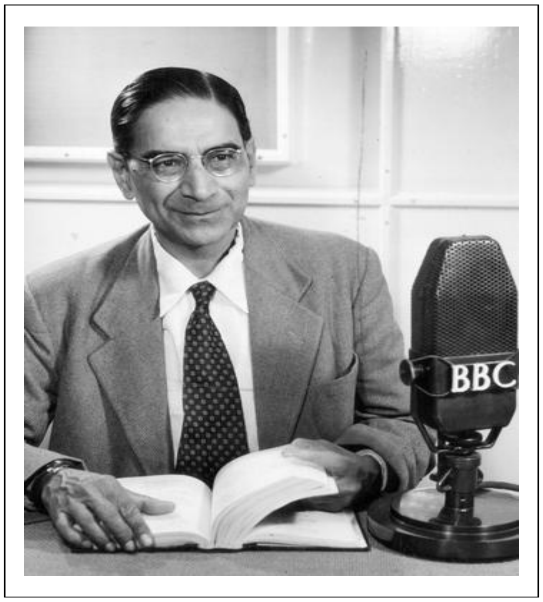

Figure 2: Visualize grayscale image of P.C. Mahalonobis in R **(R Chunk 1.1)**

So we will apply $K$**-means clustering algorithm** on our current set of vectors $V$ having $\frac{mn}{q} = 37195$ vectors each with $q = 4$ elements. And we would like to cluster these vectors in $r = 15$ many groups **(R Chunk 1.2)**.

## ► Stage - 2: (Image Compression)

Now that the prototype vectors have been determined, the next step of VQ consists of **image compression and transmission**. Each vector in the original image is compared one-at-a-time to each of the prototype vectors in the **codebook**. The prototype that most closely resembles the input vector is selected, and its address is transmitted/represented. That is the compressed version of the image is only the **codebook** consisting of $r$ vectors (instead of $\frac{mn}{q}$ vectors) each of which is $q$ dimensional.

## ► Stage - 3: (Image Decompression)

The final step of VQ consists of getting back the image, which will be obviously an approximation to the original one. i.e., a sequence of $\frac{mn}{q}$ addresses, and decompressing it. Each of the $\frac{mn}{q}$ vectors will be approximated by the corresponding prototype vector in the **codebook** (that is the cluster center which it belongs to) **(R Chunk 1.3)**.

And then by decompressing the sequence of $\frac{mn}{q}$ addresses we again reconstruct the image and visualize it **(Figure 3)**. We notice a lot of information is lost from the image in the process of lossy image compression by $K$-means algorithm.

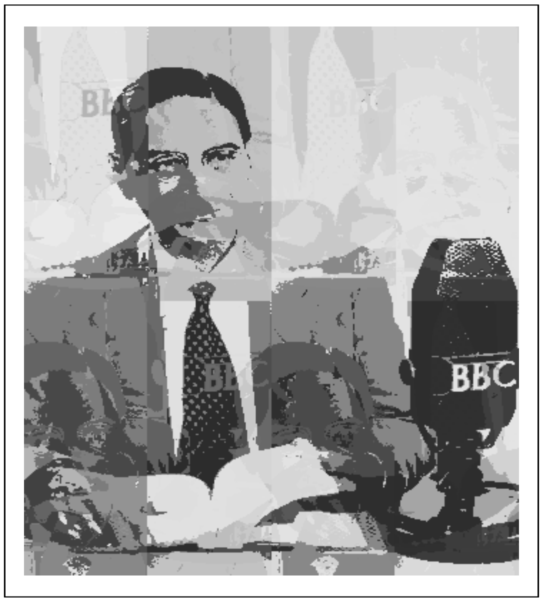

Figure 3: Decompressed and reconstructed image from codebook for $q = 4$ and $r = 15$ **(R Chunk 1.4)**.

## 2 Further Exploration

We have already implemented a version of lossy image compression by vector quantization using $K$- means clustering algorithm.

Now our job is to find suitable values of $q$ and $r$ in order to balance between loss of information and keeping codebook precised containing only relevant information from the image data. And we note that **smaller $r$ means larger compression but greater approximation**.

Now we will study how reconstructed image looks like when we compress the image by taking different values of $q$ and $r$. And also in each case we will calculate the euclidean distance **(R Chunk 2.1)** between the original image and the estimated image to observe how it changes with varying $q$ and $r$.

### ■ Case - 1:

We repeat the vector quantization (VQ) process for fixed $q = 5$ and various choices of $r$ (say $5, 3, 15, 30, 50, 100, 500$ etc.) and in each case we draw the reconstructed image from **codebook** consisting of $r$ vectors each of which is $q$ dimensional **(Figure 4)**.

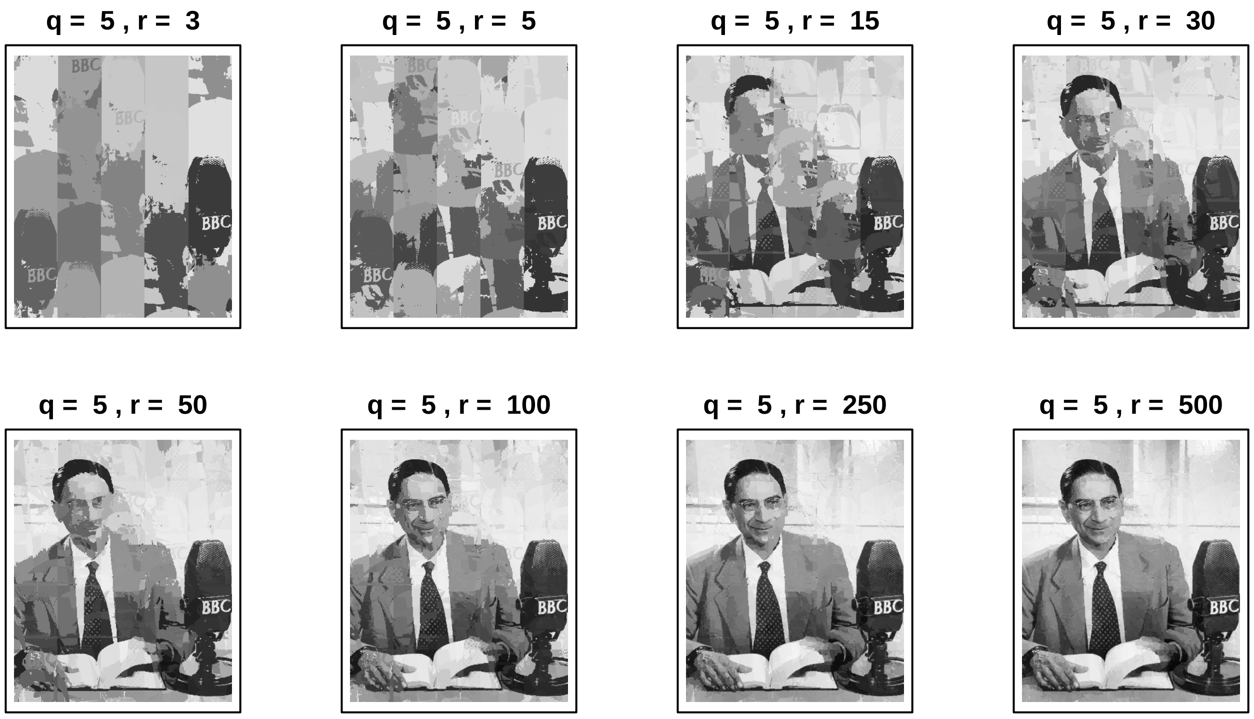

Figure 4: Reconstructed image from codebook for $q = 5$ and $r = 3, 5, 15, 30, 50, 100, 250, 500$ **(R Chunk 2.2)**.

As $r$ increases we can visually observe that image quality is improving i.e. pixels are better clustered into $r$ groups. For this we take euclidean distance as a measure of difference between original image and reconstructed image and compare the euclidean distances with fixed $q$ and varying $r$.

Now we plot the euclidean distances between the original image and the estimated image for each case i.e. fixing $q = 5$ and varying $r = 3, 5, 15, 30, 50, 100, 250, 500$ **(Figure 5)**.

Here also we see euclidean distances are decreasing as $r$ increases i.e. **larger $r$ results in less compression and less approximation**.

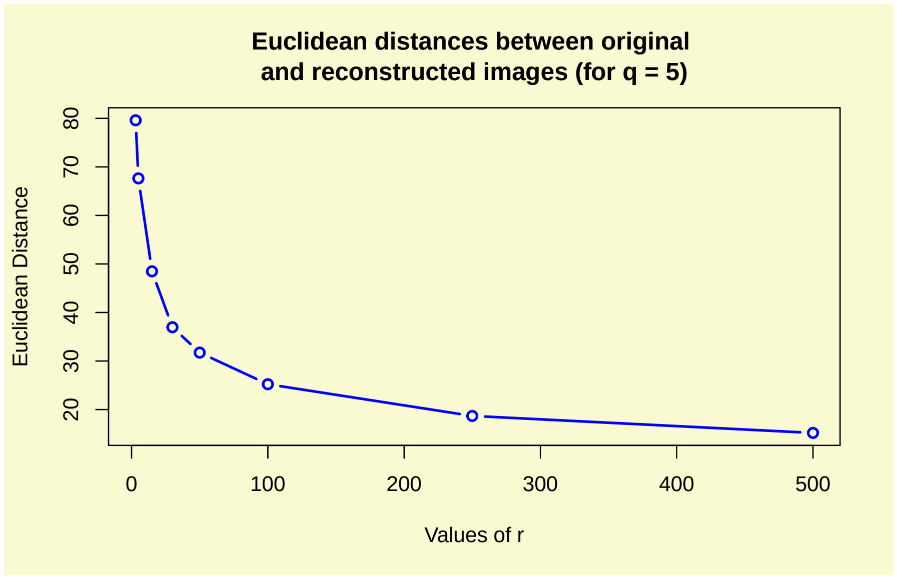

Figure 5: Euclidean distances for reconstructed images with respect to original image in case of $q = 5$ and $r = 3, 5, 15, 30, 50, 100, 250, 500$ **(R Chunk 2.3)**.

## ■ Case - 2:

We repeat the same vector quantization (VQ) process by keeping $r$ fixed (say at 15) and changing $q$ (say $2, 4, 5, 10, 15$ etc) and in each case we draw the reconstructed image from **codebook** consisting of $r$ vectors each of which is $q$ dimensional **(Figure 6)**.

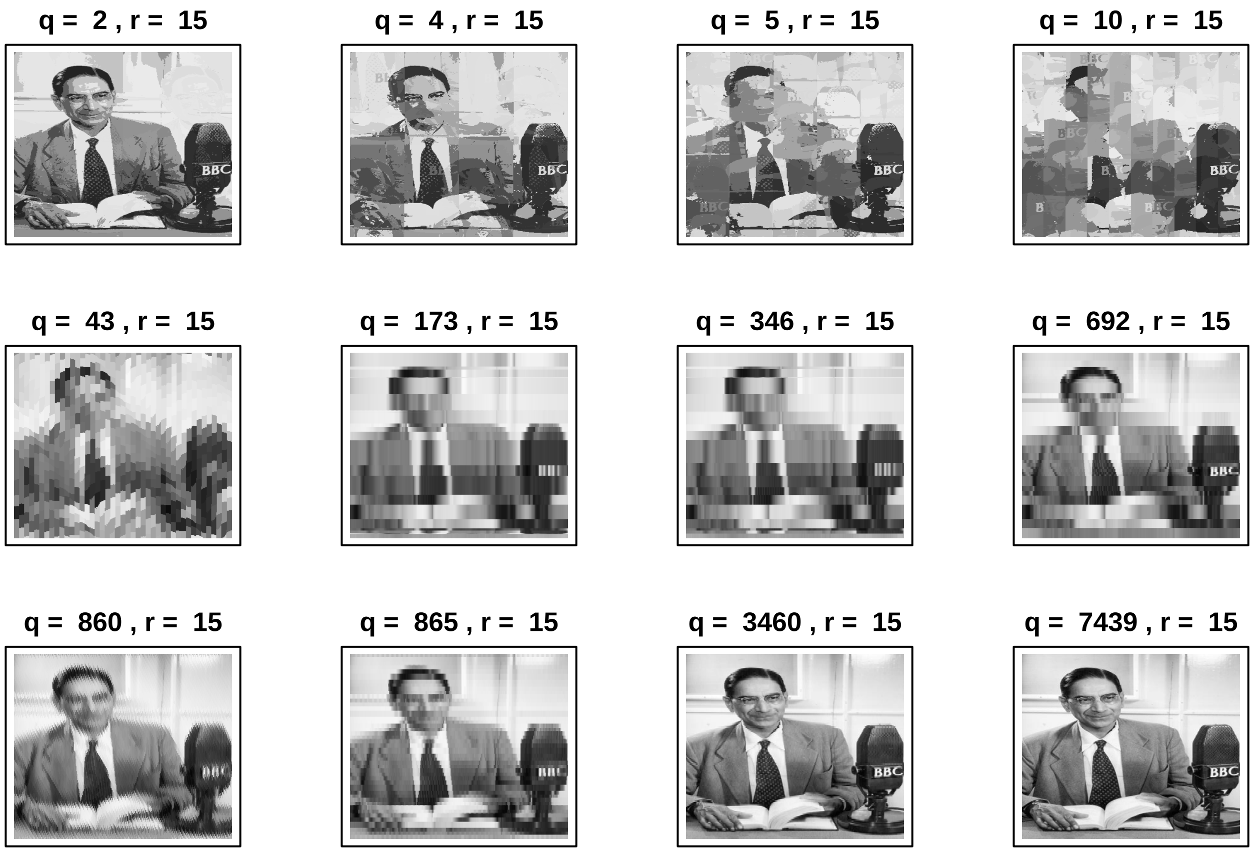

Figure 6: Reconstructed image from codebook for $r = 15$ and $q = 2, 4, 5, 10, 43, 173, 346, 692, 860, 865, 3460, 7439$ **(R Chunk 2.4)**.

As $q$ increases we can visually observe that image quality is dropping upto some $q$ and then again image quality is improving. For this we again take euclidean distance as a measure of difference between original image and reconstructed image and compare the euclidean distances with fixed $r$ and varying $q$.

Now we plot the euclidean distances between the original image and the estimated image for each case i.e. fixing $r = 15$ and varying $q = 2, 4, 5, 10, 43, 173, 346, 692, 860, 865, 3460, 7439$ **(Figure 7)**.

Here also we see euclidean distances are increasing as $q$ increases for a certain stage and then decreases as $q$ increases. This behaviour is due to the fact that when number of pixels in $V_i$ increases and when it exceeds the number of pixels present in each row of the original image then $K$- means algorithm is clustering each row pixels of the original image in addition with some more pixels of the next row pixels of the original image. That is why besides horizontal pixel approximations there is also performing vertical pixel approximations. In result the image quality derogates first and then improves periodically.

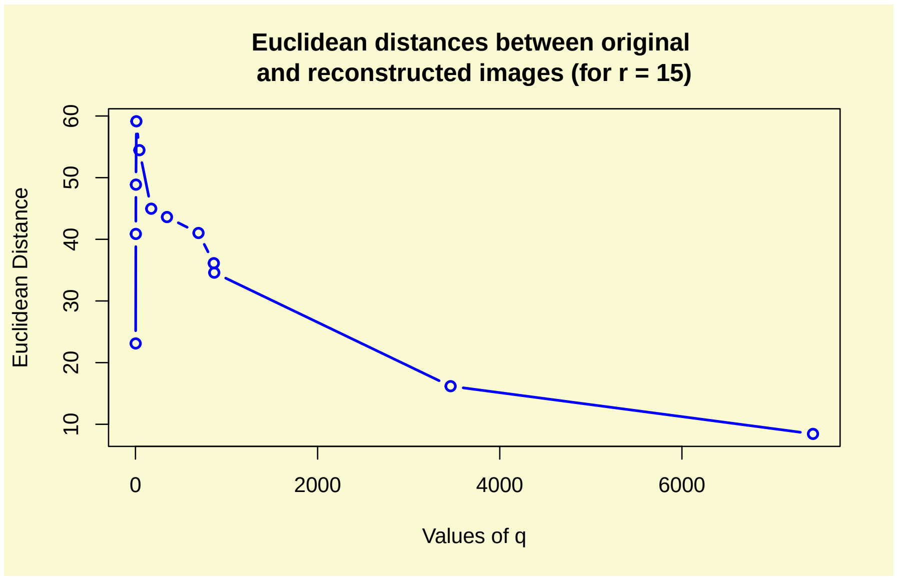

Figure 7: Euclidean distances for reconstructed images with respect to original image in case of $r = 15$ and $q = 2, 4, 5, 10, 43, 173, 346, 692, 860, 865, 3460, 7439$ **(R Chunk 2.5)**.

## R codes:

```r
# Code for: R Chunk 1.1
#install.packages("jpeg") # install "jpeg" package
library(jpeg)              # load "jpeg" package

filename = "C:/Users/ANKAN/Desktop/PG Semesters/Semester-3/AKG Multivariate/
Project 1/PCM.jpg"
image = as.vector(readJPEG(filename))

drawPic = function(y,title="")
{
        m = 430; n=346
        dim(y) = c(m, n)
        plot(1:2, ty='n', main=title, xlab="", ylab="", xaxt="na", yaxt="na")
        rasterImage(as.raster(y),1,1,2,2)
}

par(mar = c(0, 1, 0, 1))
drawPic(image)
```

```r
# Code for: R Chunk 1.2
fit_kmeans = function(image, q, r)
{
        m = 430; n=346
        dim(image) = c((m*n)/q, q)
        cl = kmeans(image, r)
        return(cl)
}

set.seed(10)
kmean = fit_kmeans(image, q=4, r=15)
```

```r
# Code for: R Chunk 1.3
reconstruct_img = function(kmean_obj, m, n){
  codebook = kmean_obj$centers
  clusters = kmean_obj$cluster
  V = codebook[clusters,]
  dim(V) = c(m, n)
  return(V)
}
```

```r
# Code for: R Chunk 1.4
reconstructed_img = reconstruct_img(kmean, m = 430, n = 346)
par(mar = c(0, 1, 0, 1))
drawPic(reconstructed_img)
```

```r
# Code for: R Chunk 2.1
euclidean_distance = function(original_image, reconstructed_image){
  original = original_image
  reconstructed = as.vector(reconstructed_image)
  d = sqrt(sum((original-reconstructed)^2))
  return(d)
}
```

```r
# Code for: R Chunk 2.2
set.seed(70)
q = 5; r_values = c(3, 5, 15, 30, 50, 100, 250, 500)
r_vs_d = vector()

par(mar = c(2, 2, 2, 2), mfrow=c(2,4))
for (r in r_values) {
  kmean = fit_kmeans(image, q, r)
  reconstructed_img = reconstruct_img(kmean, m = 430, n = 346)
  drawPic(reconstructed_img, paste("q = ", q, ", r = ", r))
  r_vs_d = c(r_vs_d, euclidean_distance(image, reconstructed_img))
}
```

```r
# Code for: R Chunk 2.3
par(mfrow=c(1,1), bg = "lightgoldenrodyellow")
plot(r_values, r_vs_d, type = "b", lwd = 2, col = "blue", xlab ="Values of r",
ylab = "Euclidean Distance", main = "Euclidean distances between original \n
and reconstructed images (for q = 5)", cex.main=1.2)
```

```r
# Code for: R Chunk 2.4
set.seed(70)
r = 15; q_values = c(2, 4, 5, 10, 43, 173, 346, 692, 860, 865, 3460, 7439)
q_vs_d = vector()

par(mar = c(2, 2, 2, 2), mfrow=c(3,4))
for (q in q_values) {
  kmean = fit_kmeans(image, q, r)
  reconstructed_img = reconstruct_img(kmean, m = 430, n = 346)
  q_vs_d = c(q_vs_d, euclidean_distance(image, reconstructed_img))
  drawPic(reconstructed_img, paste("q = ", q, ", r = ", r))
}
```

```r
# Code for: R Chunk 2.5
par(mfrow=c(1,1), bg = "lightgoldenrodyellow")
plot(q_values, q_vs_d, type = "b", lwd = 2, col = "blue", xlab ="Values of q",
ylab = "Euclidean Distance", main = "Euclidean distances between original \n
and reconstructed images (for r = 15)", cex.main=1.2)
```